

# Chapter 7

## ESTIMATION AND VELOCITY

---

In this chapter I describe the concepts of estimation and velocity. I begin with an overview of the important roles that estimation and velocity play in agile planning. I then discuss the various items that we estimate and when and how we estimate them. The bulk of the chapter focuses on how to estimate product backlog items, including how to choose a unit of measure and use Planning Poker. Next I move on to the concept of velocity and how using a velocity range is essential for planning. I discuss how new teams can forecast velocity in the absence of historical data. I conclude with ways we can influence velocity and how velocity can be misused.

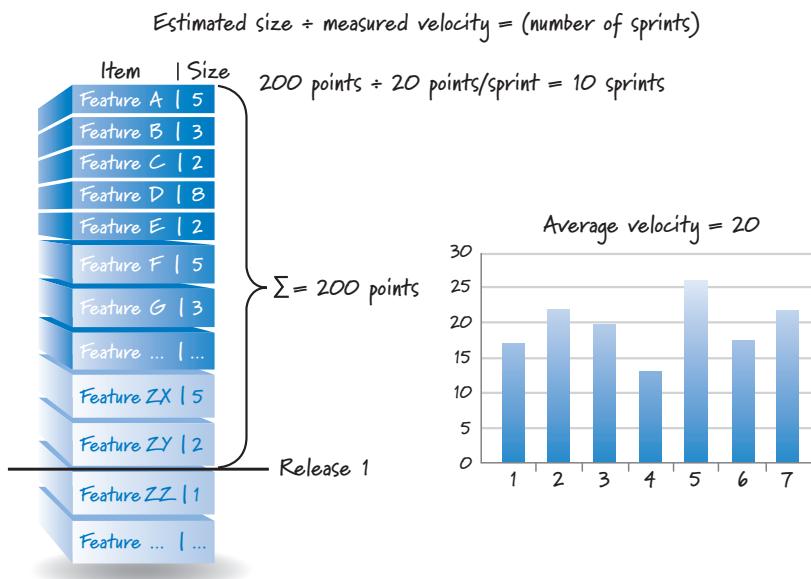
### Overview

When planning and managing the development of a product, we need to answer important questions such as “How many features will be completed?” “When will we be done?” and “How much will this cost?” To answer these questions using Scrum, we need to estimate the size of what we are building and measure the velocity or rate at which we can get work done. With that information, we can derive the likely product development duration (and the corresponding cost) by dividing the estimated size of a set of features by the team’s velocity (see Figure 7.1).

Given the product backlog in Figure 7.1, how much time do we need to create the features in Release 1? To answer that question, we must first gauge the size of Release 1. We can do this by adding the individual size estimates for each PBI targeted for Release 1. (In our example, the sum of the PBI estimates is 200 points.)

Once we know the approximate size of the release, we turn our attention to the team’s velocity, how much work the team typically gets done each sprint. Velocity is easy to measure. At the end of each sprint, we simply add the size estimates of every item that was completed during the sprint; if an item isn’t done, it doesn’t count toward velocity. The sum of the sizes of all the completed product backlog items in a sprint is the team’s velocity for that sprint. The graph in Figure 7.1 shows the team’s velocity data for the prior seven sprints. Note that the average velocity is 20.

Now that we have estimated size and measured velocity, we are in a position to calculate (derive) the duration. To do this, we simply divide the size by the velocity. If the size of Release 1 is 200 points and the team can, on average, complete 20 points of work each sprint, it should take the team 10 sprints to complete Release 1 (see



**FIGURE 7.1** The relationship among size, velocity, and duration

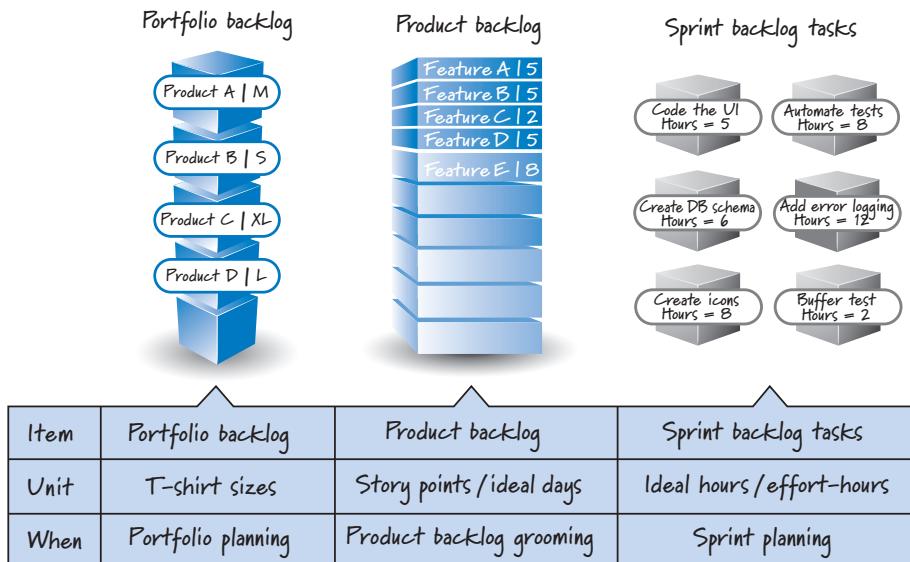
Chapter 18 for a more detailed description of release planning). Later in this chapter, I will explain why using a velocity range to do these calculations is more accurate than using an average velocity, but for illustrative purposes, I use average velocity here.

Though the basic relationship among size, velocity, and duration remains the same, some details can vary based on where you are in the development effort, what you're trying to measure, and how you intend to use the data. Let's look more closely at estimation and velocity to see how these factors change depending on what you are trying to do and when.

## What and When We Estimate

In Figure 7.1 story points were used to express the PBI estimates for calculating the release duration. Throughout the development life of a product, however, we need to estimate at varying levels of granularity and, thus, will use different units to do so (see Figure 7.2).

Most organizations make estimates for planning purposes at three different levels of detail. These estimates manifest themselves in the portfolio backlog, product backlog, and sprint backlog. Let's briefly examine each.



**FIGURE 7.2** What and when we estimate

## Portfolio Backlog Item Estimates

Although the portfolio backlog is not formally a part of Scrum, many organizations maintain one that contains a prioritized list of all of the products (or projects) that need to be built. To properly prioritize a portfolio backlog item we need to know the approximate cost of each item. As I discussed in Chapter 5, we typically won't have a complete, detailed set of requirements at the time when this cost number is initially requested, so we can't use the standard technique of estimating each individual, detailed requirement and then summing those estimates to get an aggregate estimate of the total cost.

Instead, to estimate portfolio backlog items, many organizations choose to use rough, relative size estimates like T-shirt sizes (such as small, medium, large, extra-large, and so on). I will discuss the use of T-shirt sizes for portfolio planning in Chapter 16.

## Product Backlog Estimates

Once a product or project is approved and we start adding more detail to its product backlog items, however, we need to estimate differently. When PBIs have risen in priority and been groomed to include more detail, most teams prefer to put numeric size estimates on them, using either story points or ideal days. I will discuss both of these approaches later in the chapter.

Estimating PBIs is part of the overall product backlog grooming activity. Figure 6.9 illustrates when this grooming usually takes place. Typically, PBI estimation occurs in “estimation meetings,” the first of which likely coincides with initial release planning. The product owner might also call additional estimation meetings during a sprint if any new PBIs need to be estimated.

Not all Scrum practitioners believe that PBI size estimation is a necessary activity. Their experience has shown that when Scrum teams become good enough, they are able to create PBIs that are small and of roughly the same size. Such practitioners have determined that it is wasteful to estimate small, similarly sized items. Instead, they just count the number of PBIs. They still use the concept of velocity, but it is measured as the number of PBIs that are completed in a sprint, instead of the sum of the sizes of the PBIs that are completed in a sprint.

I understand the “no-estimates-required” argument, but I still prefer to estimate PBIs for a few reasons:

- As I discussed in Chapter 5, not all PBIs will be at the same size at the same time, so there will be some larger PBIs in the backlog even if we do have a collection of smaller, similarly sized items toward the top.
- It can take some time for teams to acquire the skills to break down PBIs to be roughly the same size.
- Teams might have to split stories at unnatural points to achieve the same-size goal.
- Finally, and most importantly, one of the primary values of estimation is the learning that happens during the estimation conversations. Nothing promotes a healthy debate like asking people to put a number on something, which will immediately surface any disagreements and force assumptions to be exposed. If we were to do away with estimation, we would need to substitute an equally effective way of promoting these healthy discussions.

## Task Estimates

At the most detailed level we have the tasks that reside in the sprint backlog. Most teams choose to size their tasks during sprint planning so that they can acquire confidence that the commitments they are considering are reasonable (see Chapter 19 for more details).

Tasks are sized in **ideal hours** (also referred to as effort-hours, man-hours, or person-hours). In Figure 7.2, the team estimates that the UI task will take five effort-hours to complete. That doesn’t mean it will take five elapsed hours. It might take one person a couple of days to code the UI, or it could take a couple of people working together less than a day. The estimate simply states how much of the team’s effort is expected to complete the task. I will describe the use of task estimates in more detail in Chapter 19 when I describe the details of sprint planning.

## PBI Estimation Concepts

Though all three levels of detail are important, the remainder of this chapter focuses on product-backlog-level estimation. There are several important concepts that Scrum teams use when estimating PBIs (see Figure 7.3).

Let's examine each concept.

### Estimate as a Team

In many traditional organizations the project manager, product manager, architect, or lead developer might do the initial size estimation. Other team members might get a chance to review and comment on those estimates at a later time. In Scrum, we follow a simple rule: The people who will do the work collectively provide the estimates.

To be clear, when I say people who will do the work, I mean the development team that will do the hands-on work to design, build, and test the PBIs. The product owner and ScrumMaster don't provide estimates. Both of these roles are present when the PBIs are being estimated, but they don't do any hands-on estimation (see Figure 7.4).

The product owner's role is to describe the PBIs and to answer clarifying questions that the team might ask. The product owner should not guide or "anchor" the team toward a desired estimate. The ScrumMaster's role is to help coach and facilitate the estimation activity.

The goal is for the development team to determine the size of each PBI from its collective perspective. Because everyone sees a story from a different point of view, depending on his area of expertise, it is important that all members of the development team participate during estimation.

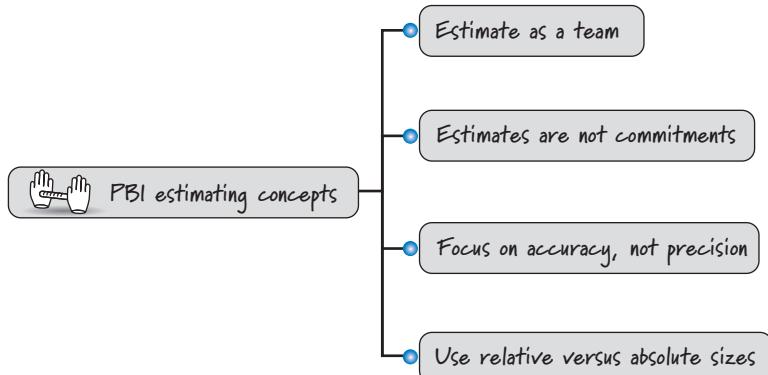
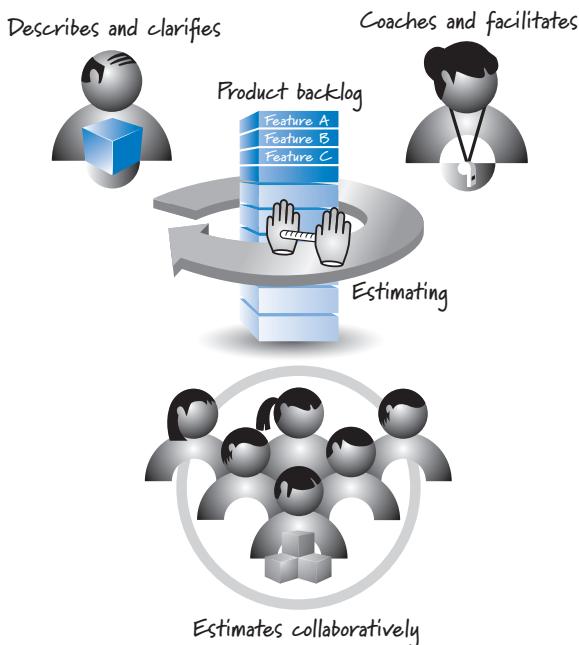


FIGURE 7.3 Product backlog item estimating concepts

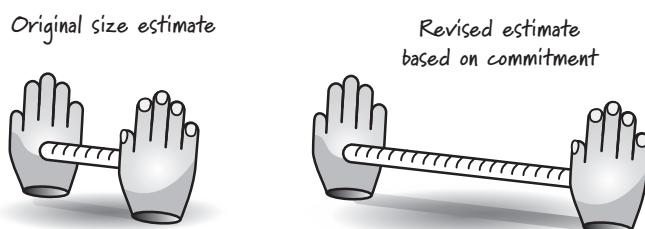


**FIGURE 7.4** The full Scrum team participates in estimation.

## Estimates Are Not Commitments

Estimates are not commitments, and it is important that we not treat them as such. That statement typically concerns managers. “What do you mean we’re not asking the team to commit to its estimates? How are we going to get precise estimates unless they do?”

When this topic comes up in my classes, I do a simple visual demonstration to make the point. I hold up a sticky note and say, “Imagine that I ask you to size this story and you tell me it’s this big.” I then use my hands to illustrate the size of story, as shown on the left side of Figure 7.5.



**FIGURE 7.5** Effect of committing on estimates

Next, I say something like “Oh, I forgot to mention, your entire bonus next year depends on your estimate being correct. I give you the opportunity to reestimate now.” At that point I start separating my hands to show progressively larger and larger estimates (see the right side of Figure 7.5). Then I usually say something like “Hey, tell me when to stop; my arms only go so far. I’m not a basketball player!”

The point is clear. If I ask people to estimate a story’s size, I expect to get a realistic estimate. If I then tell them their bonuses will be based on the estimate being correct, everyone, including me, will give a much larger estimate than the one we originally thought was correct.

The estimates ought to be a realistic measure of how big something is. We don’t want them artificially inflated due to external influences. That behavior only results in bloated schedules and a back-and-forth game of estimate inflation by team members and reduction by management. When all is said and done, we have no real understanding of the numbers because they have been manipulated so many times by different people.

## Accuracy versus Precision

Our estimates should be accurate without being overly precise. We have all been involved with products where the estimates were at a ridiculous level of **precision**. You know, the one where the estimate was 10,275 man-hours or the other one where the projected cost was \$132,865.87.

Generating these wrong, overly precise estimates is wasteful. First, there is the wasted effort of coming up with the estimate, which can be considerable. Second, there is the waste that occurs when we deceive ourselves by thinking we understand something that we don’t, and then make important, wrong, and costly business decisions based on this deception.

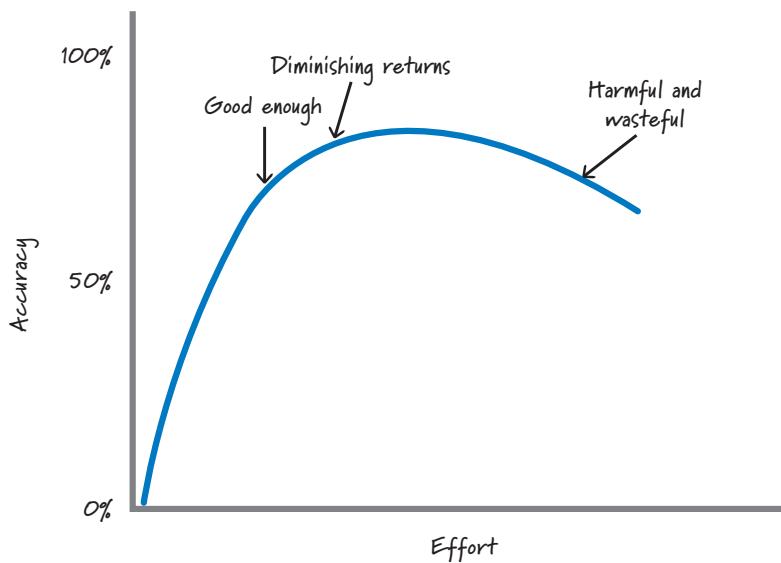
We should invest enough effort to get a good-enough, roughly right estimate (see Figure 7.6).

When estimating, there will always be a point of diminishing returns, beyond which for every additional unit of effort we invest we don’t get a corresponding increase in the **accuracy** of the estimate. Beyond that point we are just wasting our time and probably starting to negatively affect the estimate’s accuracy by considering an increasing amount of lower-value data.

## Relative Size Estimation

We should estimate PBIs using relative sizes, not absolute sizes. We compare items to determine how large an item is relative to the others (see Figure 7.7).

As shown in Figure 7.7, while it’s pretty easy to discuss how big one glass is relative to another, I might not have a good feel for the absolute quantity of liquid each glass might hold.

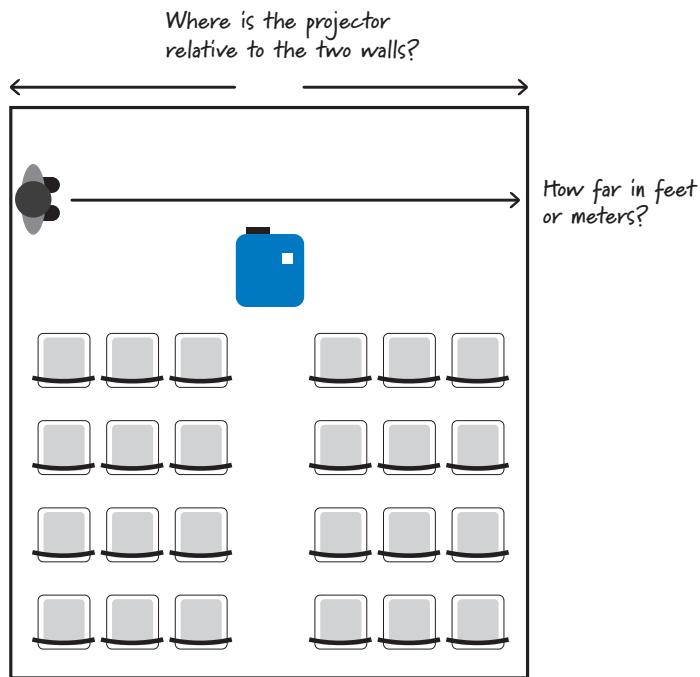


**FIGURE 7.6** Effort versus accuracy when estimating



**FIGURE 7.7** Relative size estimation

My personal observations have convinced me that people are much better at relative size estimation than absolute size estimation. Here is an example I use in my classes to illustrate the point (see Figure 7.8).



**FIGURE 7.8** Absolute versus relative size estimation

I begin by going to one side of the classroom and facing the wall on the opposite side of the room. I first ask everyone in the room to write down how far away they think I am from the opposite wall in absolute size units, for example, feet or meters. (I tell the people who look up to count ceiling tiles to stop cheating!)

In many classrooms there is typically a ceiling-mounted LCD projector that is about midway across the room. So, I then ask everyone to write down a second estimate indicating where the projector is relative to the far wall and me.

I almost always get the same results. In a typical class of 30 people, when I ask, “How far in absolute distance am I from the other wall?” I usually get 27 different answers. When I then ask, “Relative to me and the other wall, where is the projector?” 29 out of the 30 people say “about halfway”; the 30th person is just messing with me and will say something like “5/11 of the way!”

Yes, this is not a rigorous scientific experiment, but most people seem to quickly agree with the idea that they are actually better at judging relative sizes than absolute sizes. For reference, there are times when the projector is one-third of the way or two-thirds of the way across the room from me, and in those cases the results are almost always the same: most people write down the same relative size distance.

Bottom line, if we're going to ask people to estimate, we should base the technique on what people are good at (relative size estimation) and not on what they're bad at (absolute size estimation).

## PBI Estimation Units

Although there is no standard unit for PBI size estimates, by far the two most common units are story points and ideal days. There isn't a right or wrong choice when deciding between these two. I'd say 70% of the organizations I work with use story points and the other 30% use ideal days. Let's examine each.

### Story Points

Story points measure the bigness or magnitude of a PBI. We expect story points to be influenced by several factors, such as complexity and physical size. Something doesn't have to be physically large to be big. The story might represent the development of a complex business algorithm. The end result won't be very large, but the effort required to develop it might be. On the other hand, a story might be physically quite big but not complex. Let's say we have to update every cell in a 60,000-cell spreadsheet. None of the individual updates is difficult, but the updates can't be automated. How much of this work can we get done in a sprint? Though not complex, this would be a large story.

Story points combine factors like complexity and physical size into one relative size measure. The goal is to be able to compare stories and say things like "Well, if the create-a-ticket story is a 2, then the search-for-a-ticket story is an 8," implying that the searching story is roughly four times the size of the creation story.

In the example at the beginning of this chapter, the approach was to estimate the PBI sizes and then derive the duration by dividing the sum of the sizes by the average velocity. Because size measures like story points are ultimately used to calculate time (duration), story points must reflect the effort associated with the story from the development team's perspective.

### Ideal Days

An alternative approach for estimating PBIs is to use ideal days. Ideal days are a familiar unit—they represent the number of effort-days or person-days needed to complete a story. Ideal time is not the same thing as elapsed time. Ideally the American football game has four quarters that are each 15 minutes long (so the game is played in one ideal hour). However, it takes more like three to three and a half hours to actually play the game.

I stated earlier that there isn't a right or wrong answer when choosing between story points and ideal days. However, an important factor against ideal time is the risk of misinterpretation.

For example, it's currently early afternoon on Tuesday and I show you a PBI and ask, "How big is this PBI?" You say, "Two days." I say, "OK, so you'll be done Thursday early in the afternoon." You say, "No, I'm finishing up a two-day activity this afternoon and tomorrow [Wednesday]. I need the entire day just to get caught up, so I can probably start the PBI on Thursday. But since I don't have any full days to dedicate to the PBI, I'm thinking I should be done sometime next Monday." I then say, "I don't understand; you told me it was a two-day PBI, so you should be done on Thursday." You say, "I said two ideal days, not two calendar days. Please don't map my ideal days onto a calendar; it doesn't work that way."

For the 30% of the organizations that I work with that use ideal time successfully, their comment would be "Yeah, but we don't have that misinterpretation problem. We can tell people two days and they know it's not two calendar days."

If there is a low risk of misinterpretation in your organization, ideal time will likely work just fine. If you believe people will misinterpret ideal time, you're better off using story points.

There are other differences between story points and ideal time, but misinterpretation is one of the bigger issues. A student in one of my classes summed up her preference between the two when she said to her colleagues, "Look, we've been using ideal time for the past 15 years that I've been here and it has never worked. Honestly, I'd just like to try something different."

## Planning Poker

Planning Poker is a technique for sizing PBIs that was first described by James Grenning (Grenning 2002) and then popularized by Mike Cohn (Cohn 2006). Planning Poker is based on a few important concepts (see Figure 7.9).

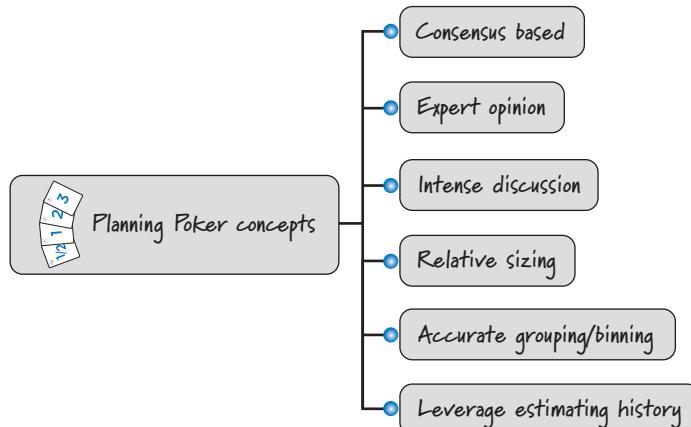


FIGURE 7.9 Planning Poker concepts

Planning Poker is a consensus-based technique for estimating effort. Knowledgeable people (the experts) slated to work on a PBI engage in an intense discussion to expose assumptions, acquire a shared understanding, and size the PBI. Planning Poker yields relative size estimates by accurately grouping or binning together items of similar size. The team leverages its established PBI estimation history to more easily estimate the next set of PBIs.

## Estimation Scale

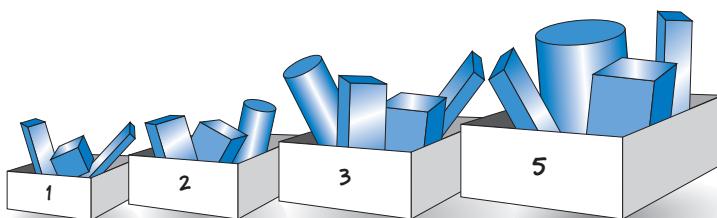
To perform Planning Poker, the team must decide which scale or sequence of numbers it will use for assigning estimates. Because our goal is to be accurate and not overly precise, we prefer to not use all of the numbers. Instead, we favor a scale of sizes with more numbers at the small end of the range and fewer, more widely spaced numbers at the large end of the range.

The most frequently used scale is the one proposed by Mike Cohn, based in part on a modified Fibonacci sequence: 1, 2, 3, 5, 8, 13, 20, 40, and 100. An alternative scale that some teams use is based on powers of 2: 1, 2, 4, 8, 16, 32, . . . .

When using this type of scale, we group or bin together like-size PBIs and assign them the same number on the scale. To illustrate this concept, let's say we work at the post office and we need to group packages of similar size together in the same bin (see Figure 7.10).

When we receive a package, we need to decide which bin to place the package in. Now, not all packages in the same bin are or will be identically the same physical shape, size, or weight, so we need to examine the packages that are currently in the bins so that we can find the best-fit bin for the package we are estimating. Once we find the closest matching bin, we put the package in the bin and move on to the next package. Obviously, the more packages we put into the bins, the easier it should be to size and bin future packages because we'll have more points of comparison.

To avoid being overly precise, we don't have a "4 bin" (if we're using a scale based on the Fibonacci sequence). So, when we get a package that we feel is larger than a 2 but smaller than an 8, we need to put it in either the "3 bin" or the "5 bin."



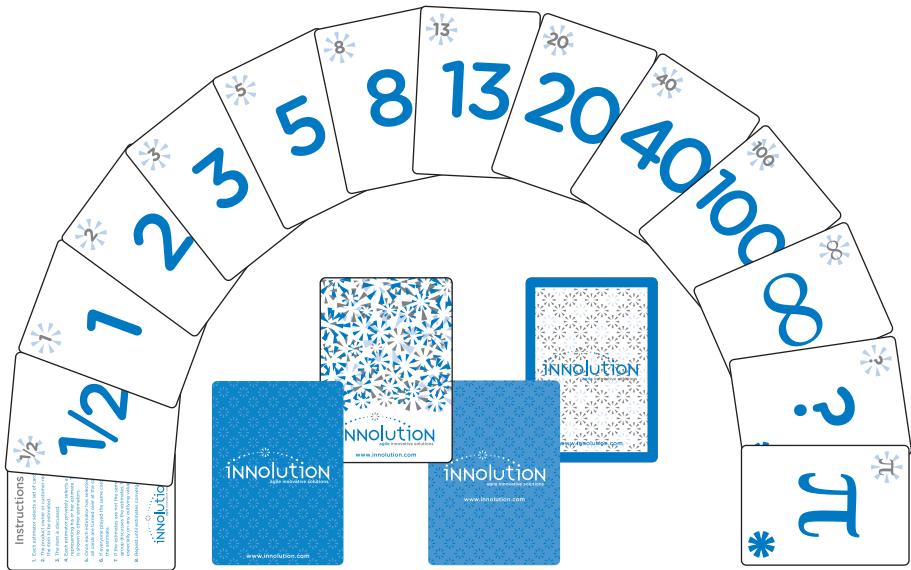
**FIGURE 7.10** Planning Poker uses binning.

## How to Play

The full Scrum team participates when performing Planning Poker. During the session, the product owner presents, describes, and clarifies PBIs. The ScrumMaster coaches the team to help it better apply Planning Poker. The ScrumMaster is also constantly looking for people who, by their body language or by their silence, seem to disagree and helping them engage. And the development team is collaboratively generating the estimates.

Each development team member is provided with a set of Planning Poker cards (see Figure 7.11).

A common interpretation of these cards is described in Table 7.1.



**FIGURE 7.11** Innolution Planning Poker cards

**TABLE 7.1** Common Interpretation of Planning Poker Cards

Card	Interpretation
0	Not shown in Figure 7.11 but included in some decks to indicate that the item is already completed or it is so small that it doesn't make sense to even give it a size number.
1/2	Used to size tiny items.
1, 2, 3	Used to size small items.

*continues*

**TABLE 7.1** Common Interpretation of Planning Poker Cards (*Continued*)

Card	Interpretation
5, 8, 13	Used to size medium items. For many teams, an item of size 13 would be the largest they would schedule into a sprint. They would break any item larger than 13 into a set of smaller items.
20, 40	Used to size large items (for example, feature- or theme-level stories).
100	Either a very large feature or an epic.
$\infty$ (infinity)	Used to indicate that the item is so large it doesn't even make sense to put a number on it.
? (question mark)	Indicates that a team member doesn't understand the item and is asking the product owner to provide additional clarification. Some team members also use the question mark as a way of recusing themselves from the estimation of the current item—typically because the person is so far removed from the item he has no idea how to estimate it. Although it is acceptable not to estimate, it is unacceptable not to participate! So, just because someone doesn't feel comfortable offering up an estimate, that doesn't allow him to disengage from the conversation or responsibility of helping the team find a consensus estimate.
$\pi$ (pi)	In this context, $\pi$ doesn't mean 3.1415926! Instead, the pi card is used when a team member wants to say, "I'm tired and hungry and I want to get some pie!" Some Planning Poker decks use a coffee cup image instead of pi. In either case, this card emphasizes an important point. The team members can engage in an intense estimation discussion for only a limited period of time (perhaps an hour or two). At that point, they really do need a break or the enthusiasm for the discussion will turn into an effort to figure out how to quickly get the estimates done, regardless of their accuracy or the learning that takes place. If people are playing the pi card, the team needs to take a break.

The rules of Planning Poker are as follows:

1. The product owner selects a PBI to be estimated and reads the item to the team.
2. Development team members discuss the item and ask clarifying questions to the product owner, who answers the questions.
3. Each estimator privately selects a card representing his estimate.
4. Once each estimator has made a private selection, all private estimates are simultaneously exposed to all estimators.
5. If everyone selects the same card, we have consensus, and that consensus number becomes the PBI estimate.

6. If the estimates are not the same, the team members engage in a focused discussion to expose assumptions and misunderstandings. Typically we start by asking the high and low estimators to explain or justify their estimates.
7. After the discussion, we return to step 3 and repeat until consensus is reached.

In Planning Poker we don't take averages or use any number not on the scale/cards. The goal is not to compromise, but instead for the development team to reach a consensus about the estimate of the story's overall size (effort) from the team perspective. Usually this consensus can be achieved within two or three rounds of voting, during which the team members' focused discussion helps obtain a shared understanding of the story.

## Benefits

Planning Poker brings together the diverse team of people who will do the work and allows them to reach consensus on an accurate estimate that is frequently much better than any one individual could produce.

As I mentioned earlier, there are some in the agile community who believe that estimating PBIs is not worthwhile. The intense discussion of the PBIs fostered by Planning Poker, however, is incredibly valuable. In my experience, you really motivate people to think about the details of the PBIs and expose any assumptions when you ask them to put a size number on them.

The majority of the value associated with Planning Poker is the discussion and better understanding that team members will share about the PBIs. I hope they also get size estimates on the PBIs; however, I am more concerned that they learn about the PBIs. If they do, they have gotten a good return on the team's investment.

## What Is Velocity?

Velocity is the amount of work completed each sprint. It is measured by adding the sizes of the PBIs that are completed by the end of the sprint. A PBI is either done or it's not done. The product owner doesn't get any value from undone items, so velocity does not include the size numbers of partially completed PBIs.

Velocity measures output (the size of what was delivered), not outcome (the value of what was delivered). We assume that if the product owner has agreed that the team should work on a PBI, it must have some value to him. However, completing a PBI of size 8 doesn't necessarily deliver more business value than completing a PBI of size 3. Perhaps the PBI of size 3 is high value and therefore we work on it early (because it is high value and low cost), and we work on the PBI of size 8 later (because it is lower value and higher cost).

Velocity is used for two important purposes. First, it is an essential concept for Scrum planning. For release-level planning, as shown in Figure 7.1, we divide the size

of the release by the team's average velocity to calculate the number of sprints necessary to complete the release. Additionally, at sprint planning, a team's velocity is used as one input to help determine its capacity to commit to work during the upcoming sprint (see Chapter 19 for more details).

Velocity is also a diagnostic metric that the team can use to evaluate and improve its use of Scrum to deliver customer value. By observing its own velocity over time, the team can gain insight into how specific process changes affect the delivery of measurable customer value.

## Calculate a Velocity Range

For planning purposes, velocity is most useful when expressed as a range, such as "The team is typically able to complete between 25 and 30 points each sprint." Using a range allows us to be accurate without being overly precise.

With a velocity range we can more accurately provide answers to questions like "When will we be done?" "How many items can we complete?" or "How much will all this cost?" Because most of these questions get asked early on in a product development effort, when we have the least information about the product, it's impossible to give a very precise answer. By using a range, we can communicate our uncertainty (see Figure 7.12).

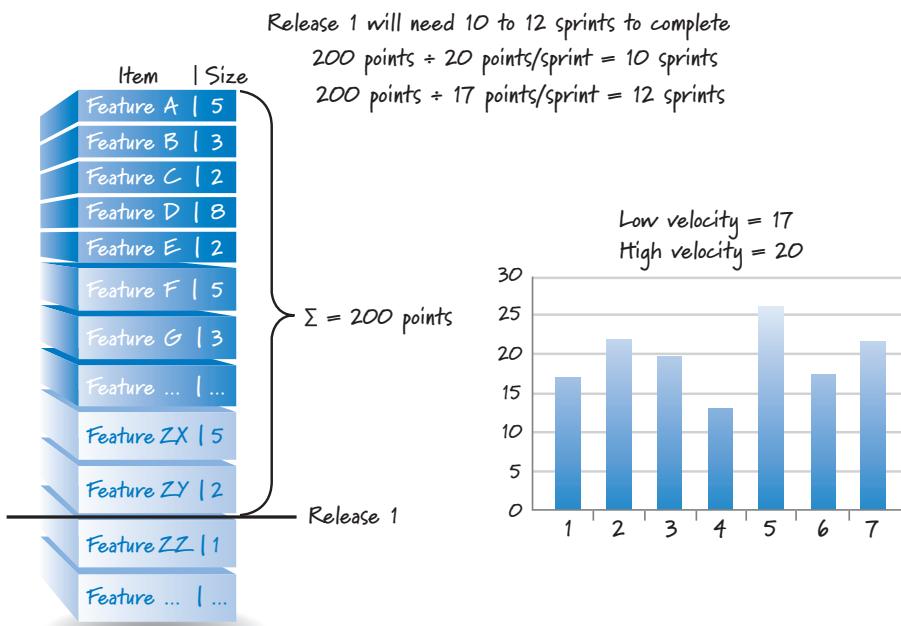


FIGURE 7.12 Calculating and using a velocity range

In this example (a revision of Figure 7.1), rather than declaring the precise sprint by which all of the items in the release will be completed (which would likely be a guess on our part), we instead provide a range as an answer to the question. To calculate this range we need two velocities for our team. If we divide the release size by the team's faster velocity, we get the fewest number of sprints required. And if we divide the release size by the team's slower velocity, we get the greatest number of sprints.

Using some simple math (like high and low averages, 90% confidence intervals, and so on), we can easily get two velocity numbers from our team's historical velocity data (17 and 20 in the example here). In Chapter 18 I will provide more detail on performing these calculations to answer questions about when, how many, and how much.

## Forecasting Velocity

In the previous examples I assumed that the team had historical velocity data that we could use to predict future velocity. Certainly one of the benefits of having long-lived teams is that they will acquire such useful historical data (see Chapter 11 for a more detailed discussion of the benefits of long-lived teams). But how do we handle the situation where we have a new team whose members haven't worked together and therefore have no historical data? We'll have to forecast it.

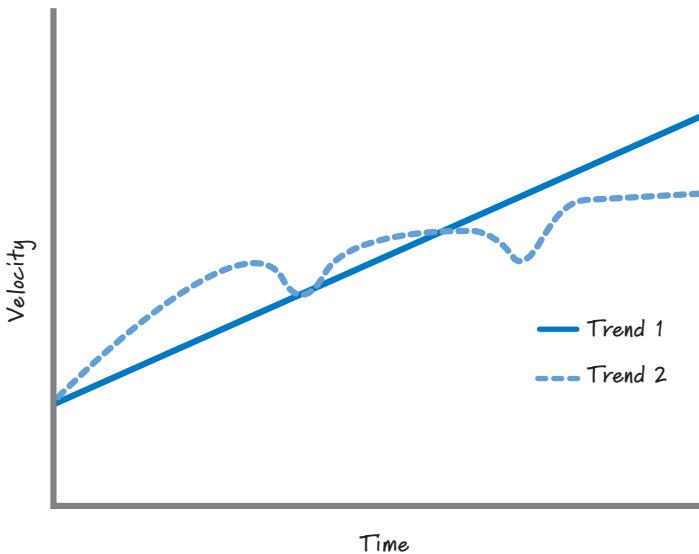
One common way to forecast a team's velocity is to have the team perform sprint planning to determine what PBIs it could commit to delivering during a single sprint. If the commitment seems reasonable, we would simply add the sizes of the committed PBIs and use that as the team's forecasted velocity.

Because what we really want is a velocity *range*, we could have the team plan two sprints and use one estimated velocity number as the high and the other as the low (the two estimates would likely be different). Alternatively, we could make some intuitive adjustments to one estimated velocity based on historical data for other teams, thereby converting the one estimate into a two-estimate range.

As soon as the team has performed a sprint and we have an actual velocity measurement, we should discard the forecast and use the actual. And as the team builds up a history of actual velocities, we should compute averages or apply other statistics to the data to extract a velocity range. (See Cohn 2009 for more examples.)

## Affecting Velocity

Do you believe that a team's velocity should constantly increase over time? An executive once said to me, "Last year my team's velocity averaged 30 points per sprint. This year I'm expecting the team to achieve 35 points per sprint." This executive believes that the team's velocity should correspond to trend 1 in Figure 7.13.



**FIGURE 7.13** A team's velocity over time

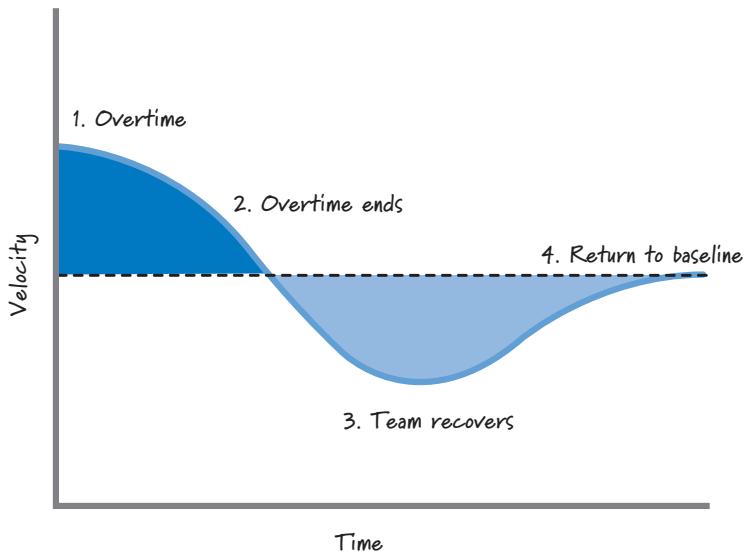
His reasoning was that if the team is constantly inspecting and adapting (continuously improving), its velocity should keep getting better and better.

I would expect a team that is aggressively trying to improve itself and is focused on delivering features in accordance with a robust definition of done and low technical debt (see Chapter 8) to see an increase in velocity. Well, at least an increase up to a certain point, at which time its velocity will likely plateau (more like trend 2 in Figure 7.13).

Just because a team's velocity has leveled out doesn't mean there is no more upward potential. There are a number of ways that the Scrum team and managers can help get velocity to the next plateau. For example, introducing new tools or increasing training can have a positive effect on velocity. Or managers can strategically change team composition with the hope that the change will eventually lead to a greater overall velocity. Of course, managers should be careful because haphazardly moving people on and off teams can and probably will cause velocity to decline.

Although introducing new tools, getting training, or changing team composition can have a positive effect on velocity, these actions usually cause a dip in velocity while the team absorbs and processes the change (see Figure 7.13, trend 2). After this decline, there will probably be an increase to the point where the team establishes a new plateau until some other change causes yet another plateau to be achievable.

Of course, there is one obvious thing we could do to try to improve velocity: work longer hours. Working a lot of consecutive overtime might initially cause velocity to increase (see "Overtime" in Figure 7.14).



**FIGURE 7.14** The effect of overtime on velocity (based on a figure from Cook 2008)

That increase will almost certainly be followed by an aggressive decline in velocity along with a simultaneous decline in quality. Even after the overtime period ends, the team will need some amount of time to recover before returning to its reasonable baseline velocity. I have seen examples of where the trough (decreased velocity area) during the recovery period is larger than the crest (increased velocity area) during the overtime period.

The end result is that lots of overtime may provide some short-term benefits, but these are frequently far outweighed by the long-term consequences.

## Misusing Velocity

Velocity is used as a planning tool and as a team diagnostic metric. It should not be used as a performance metric in an attempt to judge team productivity. When misused in this way, velocity can motivate wasteful and dangerous behavior.

For example, say I have decided to give the largest bonus to the team that has the highest velocity. Superficially this idea might seem sensible; the team with the highest velocity must be getting the most work done each sprint, right? So, why not reward that behavior?

Well, if I'm comparing teams that aren't sizing their PBIs using a common baseline (which is very likely true), comparing the numbers would make no sense. Let's say that team A assigns a value of 5 to a PBI, whereas team B assigns a value of 50 to

the same PBI. Team A doesn't really want me to compare its velocity against team B's velocity. Team B's velocity will be ten times that of team A, even if both teams actually get about the same quantity of work completed each sprint.

Once team A sees the problem, its members will start to game the system to ensure that their velocity numbers are higher. The easy way to do this is to just change the scale the team uses to estimate PBIs. So, team A now sizes the same item (the one it originally sized a 5) to be a 500. I call this behavior **point inflation**, and it serves no purpose other than to align a team's behavior with a misguided measurement system. Don't do this.

Even if teams are using the same units to consistently size PBIs, if I set up the reward system to favor bigger numbers, that's exactly what I'll get—bigger numbers (point inflation).

Even worse than point inflation is when teams cut corners to get more "done" in an effort to achieve higher, more desirable velocities. Doing so leads to increasingly greater levels of technical debt.

At the end of the day, we should judge velocity on how well it assists us with performing accurate planning and how well it helps a team to internally improve itself. Any other uses will likely promote the wrong behavior.

## Closing

In this chapter I discussed how sizes are estimated, velocity is measured, and duration is calculated. I illustrated how estimation applies to portfolio-level items, product backlog items, and tasks. I then focused specifically on PBIs by discussing important concepts related to PBI estimation, including story points and ideal days. Next I described a technique known as Planning Poker that is commonly used to estimate PBIs.

I moved from estimation to a discussion of velocity and how it should be used. I reinforced that velocity is most helpful when expressed as a range instead of a single number. I briefly mentioned ways that we might forecast the velocity for a new team. I concluded by discussing how velocity can be and frequently is misused. In the next chapter, I will focus on the concept of technical debt and how we deal with it when using Scrum.